Iptables/Netfilter savjeti i trikovi

U ovoj e-knjizi ćemo vas naučiti radu s iptables/Netfilter sustavom. Netfilter i iptables su sučelje prema linuxovoj jezgri koje omogućava *stateful* i *stateless* filtriranje mrežnog prometa, translaciju mrežnih i port adresa (NAT). Ovo sučelje je nasljednik sustava ipchains.

Netfilter i iptables se upotrebljavaju u naprednom nadzoru mreže, vatrozidima, transparentnim proxyjima i dijeljenju mrežne povezanosti (ICS - Internet Connection Sharing).

- Logirajte [1] se za dodavanje komentara

Elegantno protiv napada grubom silom na SSH

Ne tako davno pojavili su se i postali vrlo česti pokušaji neovlaštenog pristupa SSH protokolom metodom grube sile. Napadači jednostavnim skriptama, koje usmjeravaju na cijele mreže, pokušavaju pristupiti na poslužitelje iskušavanjem velikog broja korisničkih imena i/ili lozinki. Gotovo svaki dan u logovima pojave se deseci, pa i stotine zapisa o pokušajima prijavljivanja na poslužitelj s pojedine IP adrese. Tijekom napada, svake minute zlonamjernici iskušaju i po 20 imena i/ili lozinki. Slijedi jedan skraćeni primjer iz logova:

```
Mar 6 02:49:10 stroj sshd[5210]: Illegal user mat from 141.5.9.6
Mar 6 02:49:12 stroj sshd[5210]: Illegal user rolo from 141.5.9.6
Mar 6 02:49:15 stroj sshd[5273]: Illegal user ice from 141.5.9.6
Mar 6 02:49:18 stroj sshd[2827]: Illegal user horde from 141.5.9.6
Mar 6 02:49:21 stroj sshd[2689]: Illegal user cyrus from 141.5.9.6
Mar 6 02:49:24 stroj sshd[6844]: Illegal user www from 141.5.9.6
Mar 6 02:49:27 stroj sshd[3144]: Illegal user run from 141.5.9.6
Mar 6 02:49:30 stroj sshd[4021]: Illegal user matt from 141.5.9.6
Mar 6 02:49:33 stroj sshd[7440]: Illegal user test from 141.5.9.6
Mar 6 02:49:35 stroj sshd[2140]: Illegal user test from 141.5.9.6
Mar 6 02:49:38 stroj sshd[2541]: Illegal user test from 141.5.9.6
Mar 6 02:49:41 stroj sshd[8455]: Illegal user devil from 141.5.9.6
Mar 6 02:49:55 stroj sshd[2402]: Illegal user tim from 141.5.9.6
```

Ako je na vaš poslužitelj instaliran CARNetov paket kernel-cn, imate sve potrebno, i posao je biti gotov za nekoliko minuta. Dovoljno je kopirati sljedeću skriptu i pokrenuti je. Ako već imate skriptu kojom postavljate pravila iptables, onda ćete znati koji dio trebate dodati u svoju skriptu. Samo pripazite da taj dio umetnete na odgovarajuće mjesto. Slijedi skripta:

----POČETAK----

```
#!/bin/sh
set -e
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Ciscenje FILTER tabela
iptables -t filter -F
iptables -t filter -X

# Svi kojima vjerujemo idu u odgovarajucu bijelu listu
```

```
TRUSTED_SSH="161.53.0.0/16"
iptables -N SSH_WHITELIST
iptables -F SSH_WHITELIST

iptables -A SSH_WHITELIST -s $TRUSTED_SSH -m recent --remove --name SSH -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set --name SSH
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -j SSH_WHITELIST

# Rezanje pristupa i logiranje adresa s kojih dolaze SSH brute force napadi
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds
 60 --hitcount 4 \
     --rttl --name SSH -m limit --limit 2/sec -j LOG --log-prefix "SSH_brute_force:"
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds
 60 --hitcount 4 \
     --rttl --name SSH -j DROP
```

----KRAJ----

Nakon kreiranja nove ili unesenih promjena u staru, pokrećemo skriptu koja će tabele napuniti pravilima. Možemo odmah provjeriti rezultat:

```
# iptables -nL -v
```

I ne zaboravimo spremiti promjene kako bi se nova pravila učitala pri pokretanju poslužitelja:

```
# /etc/init.d/iptables save active
```

Što gornja pravila zapravo rade?

Sve navedeno omogućio nam je iptables modul "recent" (ipt_recent). Pomoću njega odredili smo, ako s neke IP adrese prema našem SSH portu 22 (--dport 22) stignu više od tri (--hitcount 4) nova (-m state --state NEW) TCP/IP paketa koji žele uspostaviti novu SSH konekciju, da se ta IP adresa blokira 60 sekundi (--seconds 60). Novi SSH paketi se ponovno počnu propuštati samo ako ih nije bilo u zadnjih 60 sekundi od zadnjeg pristiglog (--update).

Pravilom koje prethodi opisanom, logiramo samo po dva pokušaja svake sekunde (-m limit --limit 2/sec -j LOG) kako bismo spriječili DoS. Sve linije koje se zapisuju u logove imat će premetak (--log-prefix "SSH_brute_force:").

Više o opcijama koje pruža modul "recent" saznat ćete ovako:

```
# iptables -m recent --help
```

Nezaobilazni linkovi o ovoj temi su:

http://www.netfilter.org **[2]**

http://www.snowman.net/projects/ipt_recent **[3]**

- Logirajte [1] se za dodavanje komentara

sub, 2005-03-12 15:47 - Uredništvo**Kuharice:** Linux [4]
Kategorije: Sigurnost [5]
Vote: 0

No votes yet

Ipset, 1. dio

Bez zaštite poslužitelja vatrozidom (*firewallom*, ako vam je tako draže) odavno se ne može. Na mreži je previše ljudi s hakerskim sposobnostima, ali i onih koji samo misle da ih imaju. No, svi oni mogu uspješno napasti vaše poslužitelje i napraviti probleme, samo ako se odluče za to.

Iako postoje automatizirani sustavi zaštite, a ovdje prvenstveno mislimo na popularne fail2ban i OSSEC programe, nekad je pametno napraviti preventivnu zaštitu. Recimo, na poslužitelj vašeg kolege je provaljeno iz zemlje X. Iz određenih razloga pretpostavljate da će se možda pokušati provaliti i na vaše poslužitelje, pa želite spriječiti bilo koga iz te cijele zemlje da pristupi, primjerice, vašem webu.

Iako uopćeno govoreći, nije baš razborito blokirati cijele zemlje, ponekad je to jedino što možete učiniti, jer zaštita nije poznata ili moguća.

Problem nastaje u trenuku kada zaista želite blokirati neku zemlju. Ovdje se može raditi o desecima i stotinama raspona adresa, pa je problem u održavanju popisa tih adresa. Iako se za iptables može napraviti više datoteka u kojima će biti navedene adrese koje želite blokirati, nije praktično s njima baratati - većina se ipak zadovoljava standardnim *saved-active-inactive* kombincijama. Ovdje u pomoć dolazi ipset.

Pomoću ovog modula unutar iptablesa možete si olakšati način na koji rabite iptables. Možete imati osnovni vatrozid, koji brani samo ono najbitnije, a ostatak uključujete po potrebi. To izgleda ovako:

```
iptables -A INPUT -m set --set moja_pravila src -j DROP
```

Opcija -m uključuje dodatne module, primjerice "recent", "cluster", a u ovom slučaju "ipset". Da biste mogli rabiti ipset, morate ga instalirati jer ne dolazi u standardnoj instalaciji:

```
# apt-get install ipset ipset-source
...
The following extra packages will be installed:
  module-assistant
The following NEW packages will be installed:
  ipset-source module-assistant
...
Setting up ipset (2.5.0-1) ...
Setting up module-assistant (0.11.3) ...
Setting up ipset-source (2.5.0-1) ...
```

No, to nije sve. Sljedeća operacija se mora izvesti kako bi modul ipset ugradili u kernel. Restart poslužitelja nije potreban. Napravite:

```
# module-assistant auto-install ipset
```

ili jednostavnije:

```
# m-a a-i ipset

Updated infos about 1 packages
Getting source for kernel version: 2.6.32-5-686-bigmem
apt-get install linux-headers-2.6.32-5-686-bigmem
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  linux-headers-2.6.32-5-common linux-kbuild-2.6.32
The following NEW packages will be installed:
  linux-headers-2.6.32-5-686-bigmem linux-headers-2.6.32-5-common
  linux-kbuild-2.6.32

Done with /usr/src/ipset-modules-2.6.32-5-686-bigmem_2.5.0-1+2.6.32-45_i386.deb
dpkg -Ei /usr/src/ipset-modules-2.6.32-5-686-bigmem_2.5.0-1+2.6.32-45_i386.deb
Selecting previously deselected package ipset-modules-2.6.32-5-686-bigmem.
(Reading database ... 87619 files and directories currently installed.)
Unpacking ipset-modules-2.6.32-5-686-bigmem (from .../ipset-modules-2.6.32-5-68
Setting up ipset-modules-2.6.32-5-686-bigmem (2.5.0-1+2.6.32-45) ...
```

Zato što se radi o modulima unutar kernela, morali ste na ovaj način skinuti module assistant, izvorni kod kernela, modula ipset i još po koju sitnicu. No, ne morate duboko ulaziti u cijelu problematiku, dovoljno je da slijedite upute koje smo ovdje naveli.

Dobro, kako dalje? Kako se uopće prave setovi? Ništa lakše:

```
ipset --create moja_pravila nethash
ipset --add moja_pravila 109.228.64.0/18
ipset --add moja_pravila 79.140.144.0/20
ipset --add moja_pravila 195.66.160.0/19
ipset --add moja_pravila 85.94.96.0/19
ipset --add moja_pravila 77.222.0.0/19
ipset --add moja_pravila 78.155.32.0/19
ipset --add moja_pravila 95.155.0.0/18
ipset --add moja_pravila 109.228.64.0/18
ipset --add moja_pravila 46.33.192.0/19
...
```

U navedenom popisu mogu biti na stotine, pa i tisuće adresa. Pazite da to budu mrežne adrese, odnosno rasponi, ne miješajte ovdje i adrese hostova. Odnosno, probajte, dobit ćete:

```
ipset v2.5.0: Out of range cidr `109.228.113.111/32' specified
```

Sada kada ste napravili set pravila "moja_pravila", vrlo je jednostavno ubaciti taj set u iptables:

```
# iptables -A INPUT -m set --set moja_pravila src -j DROP
# iptables -L -n  | grep moja_pravila
DROP       all  --  0.0.0.0/0            0.0.0.0/0           match-
set   moja_pravila src
```

S jednim retkom koda blokirali ste tisuće adresa, na elegantan i brz način. U primjeru smo rabili zastavicu "**src**", što naravno određuje da se pravila primjenjuju na dolazne adrese. Analogno, zastavica "**dst**" određuje da se radi o odredišnim adresama.

Dobro, a po čemu je to bolje od standardnog načina, to sve mogu i preko "običnih" iptables pravila, pitate se.

Iptables su moćna stvar, ali je rukovanje s velikim brojem pravila je nespretno, jer morate paziti na njihov poredak koji je jako bitan. Zbog toga može doći do neželjnih kombinacija, što može dopustiti vanjskim adresama više nego što ste htjeli, odnosno može čak i međusobno poništiti vašu prvobitnu namjeru. O tome u nastavku, gdje ćemo navesti i neke dodatne primjere koji će vam olakšati uporabu ipset modula.

- Logirajte [1] se za dodavanje komentara

pet, 2012-09-28 11:32 - Željko Borš**Kuharice:** Linux [4]
Kategorije: Software [6]
Operacijski sustavi [7]
Servisi [8]
Vote: 5

Vaša ocjena: Nema Average: 5 (1 vote)

Filtriranje mrežnog prometa na aplikacijskoj razini pomoću Linux rješenja - 1.dio

Uvod
Mrežni filtar tj. firewall postao je dio "mrežne svakodnevice" sredinom 80-tih godina prošlog stoljeća, još od pojave usmjernika (routera). Velika većina firewalla ima mogućnost filtriranja mrežnog prometa isključivo na drugom, trećem i četvrtom OSI sloju (podatkovni, mrežni i prijenosni slojevi), tj. prema MAC adresi, IP adresi, priključnoj točki (portu usluge) i stanju konekcije. Sve masovnija upotreba računalnih mreža dovodi do potrebe kontroliranja mrežnog prometa na sedmom sloju tj. aplikacijskoj razini.

Mrežni se promet filtrira prema protokolima koji se koriste na ovoj razini OSI modela, pa čak i prema samim vrstama datoteka koje se prenose mrežom. Uz komercijalne proizvode (npr. Check Point VPN-1) pojavila su se i rješenja bazirana na open-source tehnologijama, a jedno takvo rješenje objašnjeno je u ovom članku.

Kako ovaj postupak zahtijeva "patchiranje" i (pre)kompajliranje kernela, preporuka je da ovo ne pokušavate na produkcijskom poslužitelju, nego da za ovu namjenu izdvojite zasebno računalo, koje ne mora biti najnovije. Tako ulogu mrežnog filtra i DHCP poslužitelja za otprilike 50-ak računala, u

mom slučaju, sasvim zadovoljavajuće obavlja Pentium III na 800MHz s 256MB radne memorije, tvrdim diskom od 20GB i dvije mrežne kartice.

Za početak, potrebno je ukratko objasniti nama interesantan način rada mrežne kartice. Red čekanja (*queue*) je privremena memorija (*buffer*) ili lokacija, koja sadrži konačan broj paketa, koji čekaju da se nad njima obavi neka akcija. Svako mrežno sučelje ima svoj predefinirani red čekanja, a u njemu postoje 3 moguće akcije: ulazak paketa u red (*enqueue*), njegov izlazak iz reda (*dequeue*) i njegovo brisanje (*drop*). U najjednostavnijem se obliku paketi u redu čekanja prosljeđuju po FIFO principu i bez drugih mehanizama nije moguće kontrolirati njihovo ponašanje. Metode upravljanja redovima (*queuing disciplines - qdiscs*) su algoritmi kojima se kontrolira način ulaska paketa u redove i njihovog izlaska. Ukratko, potrebno je prepoznati željeni protokol tj. paket koji pripada interesantnom toku podataka (*match*), nekako ga označiti (*mark*) i onda ga na neki način oblikovati (*shape*). Više o tome možete pročitati u Linux Advanced Routing and Traffic Control HOWTO dokumentu (lartc.org [9]).

Instalacija
Sama instalacija nije komplicirana, a najviše vremena oduzima (pre)kompajliranje kernela.
Potrebni paketi na računalu su:
- 2.4 ili 2.6 kernel source (kernel.org [10]) - preferirano 2.6
- iptables [11] source
- iproute2 [12]
- l7-filter [13]
- definicije protokola [14]
Napomena: trenutna verzija l7 filtra nije kompatibilna s kernel verzijom 2.6.20.x.

Za potrebe ovog članka svi paketi skinuti su u direktorij /usr/src/.

Da bi uključili podršku za l7 filtar, potrebno je patchirati kernel source i iptables. Patchevi za odgovarajuće verzije nalaze se unutar l7-filter paketa, pa prvo raspakiramo paket l7-filter u proizvoljni direktorij (u primjeru je korišten direktorij /usr/src/netfilter-layer7):

```
tar xzvf netfilter-layer7-vX.Y.tar.gz /root/netfilter-l7
```

Nakon što smo nabavili kernel source, raspakiramo ga unutar /usr/src direktorija i pozicioniramo se u dobiveni direktorij /usr/src/linux-2.6.X.Y:

```
tar xzvf linux-2.6.X.Y.tar.gz

cd linux-2.6.X.Y
```

Zatim je potrebno patchirati kernel source:

```
patch -p1 < /usr/src/netfilter-layer7/kernel-2.6.X-layer7-Y.patch
```

Nakon toga, slijedi podešavanje kernela po želji (npr. pomoću make menuconfig metode - lokacije vrijede za verziju 2.6.18), uz obavezno uključenje sljedećih opcija:
- "Prompt for development and/or incomplete code/drivers" (pod "Code maturity level options")
- "Network packet filtering" (Networking › Networking options › Network packet filtering)
- "Netfilter Xtables support" (Networking › Networking options ›Network packet filtering › Core Netfilter Configuration)
- "Connection tracking" (Networking › Networking options › Network packet filtering › IP: Netfilter Configuration)
- "Connection tracking flow accounting" (u istom izborniku kao i prethodna opcija)
- "IP tables support" (u istom izborniku)
- "Layer 7 match support" (u istom izborniku)
Ostale Netfilter opcije nisu nužne, ali su poželjne (naročito FTP support).

Slijedi uobičajeno kompajliranje i instalacija kernela te restart računala:

```
make all

make modules_install

make install

mkinitrd -o /boot/initrd.img-2.6.18 2.6.18
```

uz podešavanje boot loadera (lilo/grub ili nešto treće).

Na redu je iptables. Prvo otpakiramo paket (npr. u /usr/src/iptables):

```
tar xzvf iptables-X.Y /usr/src/iptables

cd iptables
```

i zatim patchiramo iptables source:

```
patch -p1 < /usr/src/ netfilter-layer7/iptables-layer7-2.x.patch
```

Još je potrebno dodati pravo izvršavanja na datoteku ".layer7-test" unutar /root/iptables/extension direktorija:

```
chmod +x extensions/.layer7-test
```

Slijedi kompajliranje iptables-a:

```
make KERNEL_DIR=/putanja/do/patchiranog/kernela (u našem slu?aju /usr/src/linux-2.6.X
.Y)
```

i zatim instalacija (kao root):

```
make install KERNEL_DIR=/putanja/do/patchiranog/kernela (u našem slu?aju /usr/src/lin
ux-2.6.X.Y)
```

Za uspješnu instalaciju potrebno je imati već patchirani i konfigurirani kernel source.

Slijedi postavljanje definicija protokola koje je najbolje staviti u /etc/l7-protocols direktorij:

```
tar xzvf l7-protocols-YYYY-MM-DD.tar.gz /etc/l7-protocols
```

Moguća je njihova instalacija u proizvoljni direktorij, ali je za korištenje istih potrebno navesti njihovu lokaciju sa opcijom --l7dir.

Iz razloga što na paketu iproute2 nisu potrebne nikakve intervencije, dovoljno ih je instalirati:

```
apt-get update
```

```
apt-get install iproute
```

Upotreba
Sad ste spremni za rad. Moguće radnje su: blokiranje određenih protokola, kontroliranje pojasne širine i praćenje stanja na mreži. Svaka navedena radnja bit će opisana dalje u tekstu.

l7-filtar koristi standardnu iptables sintaksu, a osnovna sintaksa glasi:

```
iptables [tablica i lanac] -m layer7 --l7proto [ime_protokola] -j [akcija]
```

Iptables sintaksu možete naći na netfilter.org [11].

l7-filtar treba "vidjeti" sav mrežni promet koji želimo kontrolirati, što znači da promet treba proći pravila l7-filtra. To se postiže upotrebom POSTROUTING lanca u mangle tablici:

```
iptables -t mangle -A POSTROUTING -m layer7 --l7proto [itd.]
```

Napomena: ukoliko se koristi l7-filter verzija starija od 2.7, potrebno je ručno učitati ip_conntrack modul za kernel da bi l7-filtar ispravno radio. Novije verzije ga učitaju automatski.

Blokiranje
Blokiranje nije najpoželjniji način kontrole mrežnog prometa, i to iz više razloga:
- l7-filtar "matching" nije neotporan, tj. može se dogoditi da jedan protokol izgleda kao drugi (*false positive*)
- skoro svaka vrsta mrežnog prometa je legitimna (primjer su P2P protokoli koji se koriste za legalno razmjenjivanje i distribuciju besplatnih i slobodnih programa i dokumenata, a istovremeno se koriste za masovno kršenje autorskih prava)
Treba imati na umu da l7-filtar nije dizajniran s namjerom da se mrežni promet blokira, pa bi ovu radnju trebalo koristiti samo u nuždi.

Kontrola pojasne širine
Za kontrolu pojasne širine koristi se Netfilter za označavanje paketa (*mark*) i zatim se pomoću QoS (*Quality of Service*) tehnika može oblikovati promet označenih paketa. Samo označavanje radi se s opcijom

```
-j MARK --set-mark [integer]
```

dok se za oblikovanje koristi tc komandna linija koja je dio IPROUTE paketa. Slijedi primjer označavanja i filtriranja paketa koji koristi imap protokol:

```
iptables -t mangle -A POSTROUTING -m layer7 --l7proto imap -j MARK --set-mark 3
```

Vrijednost [integer] varijable je proizvoljna.

Oblikovanje mrežnog prometa tog označenog paketa se može izvoditi na sljedeći način:

```
tc filter add dev eth0 protocol ip parent 1:0 prio 1 handle 3 fw flowid 1:3
```

Time smo označeni imap promet usmjerili na treću podklasu prio metode za upravljanje redom čekanja (više o metodama za upravljanje slijedi kasnije u članku).
Komplicirana i nerazumljiva sintaksa tc komandne linije opisana je u LARTC HOWTO dokumentu, a za konkretnu upotrebu i lakši početak, u prilogu članka su dvije gotove skripte za oblikovanje mrežnog prometa. Jedna je skripta za premosnike (bridges), a jedna za "ne-premosnike" (non-bridges). Skriptu je potrebno modificirati na način da se odredi protokol koji se želi pratiti tj. oblikovati. Također, u

slučaju ne-premosnika, potrebno je konfigurirati i NAT servis, a primjer NAT skripte je u prilogu. Skriptu treba modificirati na način da se promjeni IP adresa na kojoj se nalazi računalo koje obavlja NAT. Preporučljivo je da se skripte stave unutar /etc/init.d/ direktorija te se stave u startup proceduru sustava naredbom:

```
update-rc.d -f <ime_skripte> defaults
```

Podržan je priličan broj protokola, a to su uglavnom (nepoželjni) P2P protokoli te protokoli koje koriste računalne igre. Također je moguće napisati definicije za nove protokole, ukoliko za to postoji potreba. Uz protokole, moguće je definirati i tipove datoteka čiji promet želimo kontrolirati. Tako su podržane datoteke exe, gif, jpeg, pdf, rar, zip itd. Listu podržanih protokola i tipova datoteka možete naći pri kraju članka.

Praćenje prometa na mreži
Ako vas samo zanima kojim se intenzitetom koriste protokoli koje ste definirali unutar prve skripte, moguće je koristiti gornju naredbu, ali bez -j opcije. Na primjer:

```
iptables -t mangle -A POSTROUTING -m layer7 --l7proto imap
```

Statistika se u tom slučaju može pratiti pomoću naredbe

```
iptables  -t mangle -L -v
```

Već smo rekli da su metode upravljanja redovima algoritmi kojima se kontrolira način ulaska paketa u red i njihov izlazak. Ti algoritmi uključuju odlučivanje o tome koji se paketi propuštaju, kojim redoslijedom i brzinom, a to se radi unošenjem kašnjenja, preraspodjelom redoslijeda paketa i njihovim prioritiziranjem. Naravno, postoji više vrsta metoda za upravljanje, a u sljedećih nekoliko članaka sistematizirat ću osnovne metode upravljanja redovima čekanja i time olakšati odabir odgovarajuće metode ili više njih.

Prilozi:
Skripta za premosnike (bridges) [15]
Skripta za ne-premosnike (non-bridges) [16]
Skripta za NAT [17]

Linkovi:
l7-filter home page [18]
podržani protokoli i tipovi datoteka [19]
Linux Advanced Routing and Traffic Control [9]

- Logirajte [1] se za dodavanje komentara

čet, 2007-05-17 14:59 - Mirko Lovričević**Kuharice:** Za sistemce [20]
Kategorije: Mreža [21]
Vote: 0

No votes yet

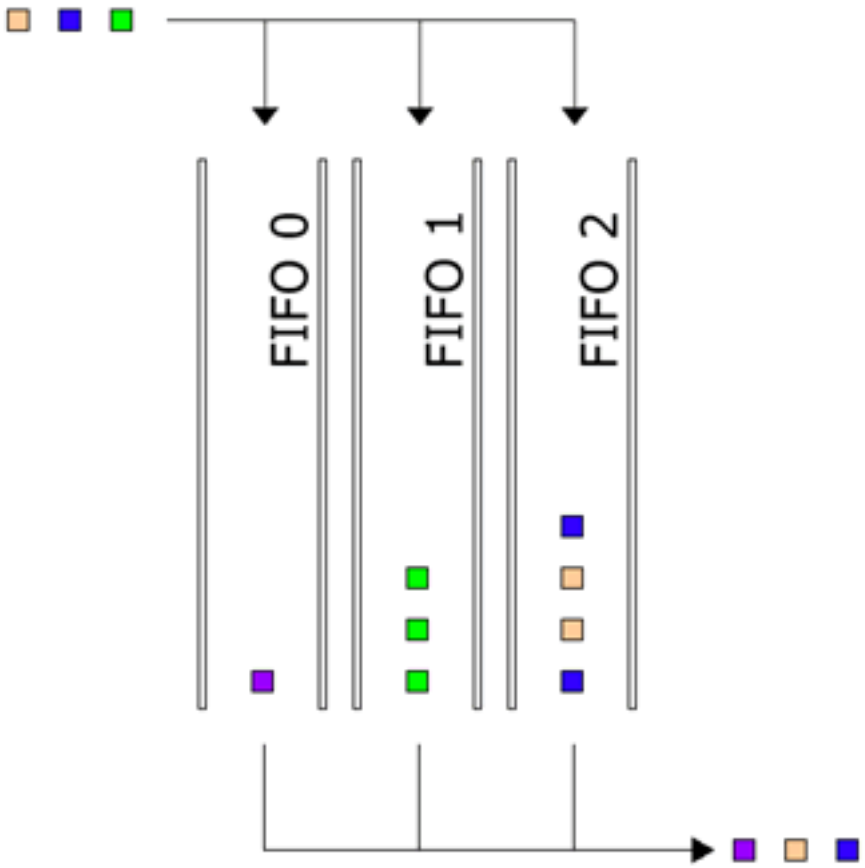
Filtriranje mrežnog prometa na aplikacijskoj razini pomoću Linux rješenja - 2.dio

U ovom članku sistematizirat ću osnovne metode upravljanja redovima čekanja na Linux operativnim sustavima, a to su besklasne metode upravljanja redovima.

Red čekanja (*queue*) je privremena memorija (*buffer*) ili lokacija, koja sadrži konačan broj paketa, koji čekaju da se nad njima obavi neka akcija. Svako mrežno sučelje ima svoj predefinirani red čekanja, a u njemu postoje 3 moguće akcije: ulazak paketa u red (*enqueue*), njegov izlazak iz reda (*dequeue*) i njegovo brisanje (*drop*). U najjednostavnijem se obliku paketi u redu čekanja prosljeđuju po FIFO principu i bez drugih mehanizama nije moguće kontrolirati njihovo ponašanje. Metode upravljanja redovima (*queuing disciplines - qdiscs*) su algoritmi kojima se kontrolira način ulaska paketa u redove i njihovog izlaska. Ukratko, potrebno je prepoznati željeni protokol tj. paket koji pripada interesantnom toku podataka (*match*), nekako ga označiti (*mark*) i onda ga na neki način oblikovati (*shape*).

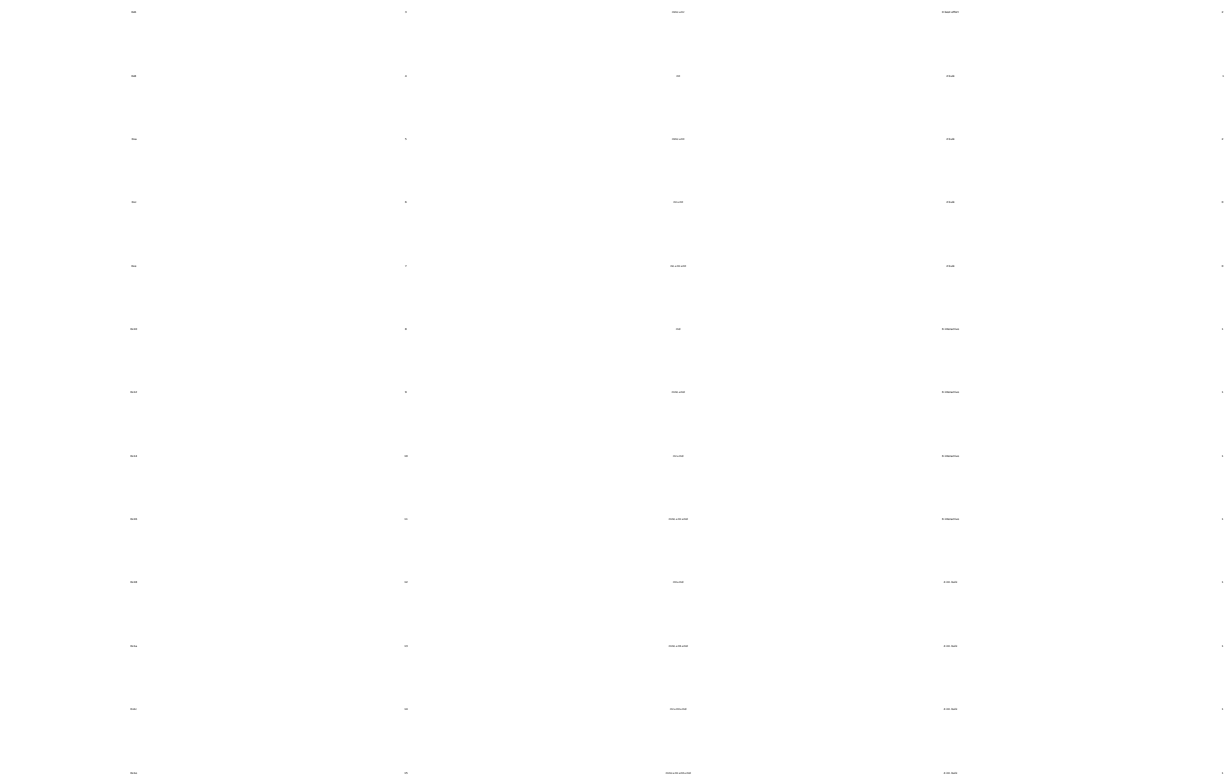
Besklasne metode za upravljanje
Besklasne metode (*classless qdiscs*) su osnovni mehanizmi upravljanja redovima čekanja na Linux operativnim sustavima i njiha se isključivo može kontrolirati izlazni (odlazni) mrežni promet. Te metode ne mogu sadržavati klase i njima nije moguće dodijeliti filtar. Pomoću ovih metoda moguće je oblikovati mrežni promet isključivo na cjelokupnom mrežnom sučelju, bez ikakve njegove podjele, a sama kontrola prometa se obavlja preraspodjelom paketa, unošenjem kašnjenja i brisanjem paketa.

Metoda pfifo_fast
Ovo je osnovna metoda upravljanja redom i bazirana je na first-in first-out (FIFO) principu, a ujedno je i predefinirana metoda upravljanja redom na Linuxu. Za razliku od bazne FIFO metode, pfifo_fast metoda ima mogućnost davanja prioriteta paketima na način da je glavni red podjeljen u tri pod reda u kojima vrijedi FIFO princip i svaki od njih ima svoj prioritet. Shema pfifo_fast metode je prikazana na slici 1.



Važno je naglasiti da pod red 0 ima najveći prioritet, a pod red 2 najmanji. Zbog toga se pod red 1 neće početi prazniti sve dok postoje paketi koji čekaju u pod redu 0. Isto vrijedi i za pod red 2, on neće biti ispražnjen sve dok postoje paketi u pod redu 1.
Kako je pfifo_fast metoda predefinirana i na njenu konfiguraciju nije moguće utjecati, ne postoje ni parametri za njeno podešavanje. Međutim, postoje dva parametra vezana uz ovu metodu, ali nijedan od njih nije moguće postavljati pomoću *tc* komandne linije. Prvi parametar jest *priomap* koji određuje prioritet paketa. Jezgra operativnog sustava čita TOS (Type of Service) polje u zaglavlju IP paketa i prema vrijednosti tog polja paketi se razvrstavaju u pod redove. Četiri od osam bitova TOS polja su definirana na način prikazan u tablici 1.

Linux operativni sustavi imaju predefiniranu interpretaciju svih kombinacija vrijednosti TOS polja i preslikavanje u pod redove na način prikazan u tablici 2.



Stupac "Bitovi" sadrži vrijednosti četiri TOS bita, a stupac "Značenje" Linux interpretaciju vrijednosti polja "Bitovi". Na primjer, vrijednost 15 u stupcu "Bitovi" znači da paket traži minimiziranje troška, maksimalnu pouzdanost, maksimalnu propusnost i minimalno kašnjenje, a po interpretaciji te vrijednosti, paket će biti preslikan u pod red 1.
Ovaj parametar također omogućuje postavljanje većeg prioriteta koji nisu povezani s TOS poljem u zaglavlju paketa, nego se postavljaju pomoću drugih mehanizama, a načini su opisani u RFC 1349.

Drugi parametar jest *txqueuelen*, a koji određuje duljinu reda. Njega je moguće postaviti prilikom konfiguracije mrežnog sučelja, npr. sa

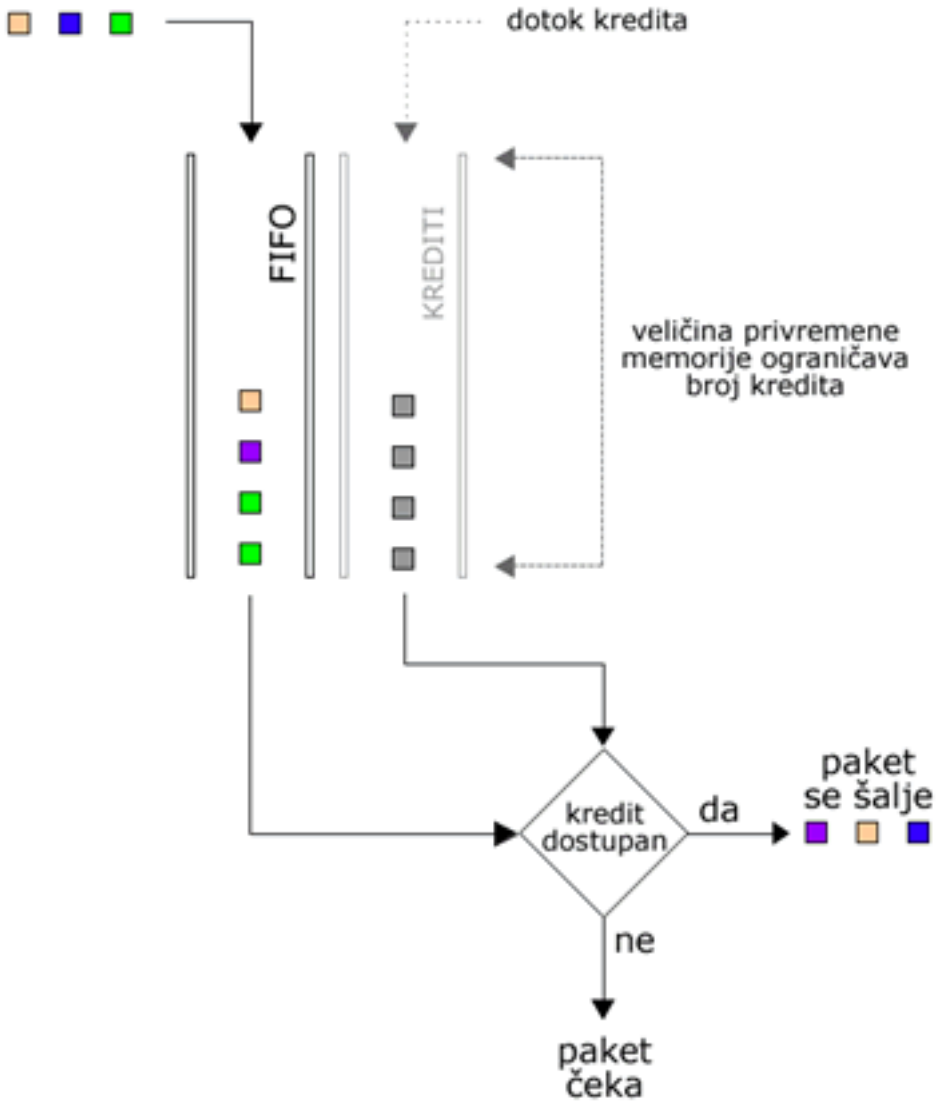
```
#ifconfig eth0 txqueuelen 10
```

Njegova trenutna vrijednost se može se provjeriti s *ifconfig* i *ip* naredbama.

Metoda filtra s kreditima
Metoda filtra s kreditima (Token Bucket Filter – TBF) je jednostavna metoda za upravljanje redom koja radi na način da propušta samo one pakete koji pristižu na mrežno sučelje ne većom brzinom od određene, uz mogućnost kratkotrajnih prekoračenja (*burst*). Ova je metoda prilično precizna i koristi malo procesorskog vremena tako da bi trebala biti prvi izbor u slučaju da se mrežno sučelje želi samo usporiti. Implementacija TBF metode sastoji se od privremene memorije (*bucket*) koja se konstantno puni imaginarnim podacima - kreditima za slanje (*token*) nekom određenom brzinom (*token rate*). Najvažniji parametar je veličina privremene memorije tj. broj imaginarnih podataka koji mogu biti pohranjeni u nju. Svaki kredit koji pristigne pokupi jedan dolazeći paket iz toka podataka i onda se briše iz privremene memorije. Povezujući ovaj algoritam s dva toka - podataka i kredita, dolazi se do tri moguće kombinacije:
- podaci stižu u TBF jednakom brzinom kao i krediti - u ovom slučaju svaki dolazeći paket ima svoj odgovarajući kredit i paket prolazi bez zadržavanja
- podaci stižu i TBF brzinom manjom od brzine kredita - samo dio kredita pokupi dolazeće pakete i krediti se akumuliraju do veličine privremene memorije - akumulirani
se krediti mogu iskoristiti za kratkotrajno slanje paketa brzinom većom od standardne brzine dolaska kredita (*burst*)
- podaci stižu u TBF brzinom većom od brzine kredita - ovo znači da će privremena memorija vrlo brzo ostati bez kredita i paketi ostaju u redu čekanja - u tom slučaju se

TBF počinje gušiti i ako paketi nastave stizati brzinom većom od brzine kredita, dolazeći paketi će se brisati (*drop*)

Shema TBF metode je prikazana na slici 2.



U samoj implementaciji krediti predstavljaju oktete, a ne pakete.
Tri parametra koja su uvijek dostupna za upotrebu uz TBF (neovisno o količini slobodnih krediti) su:
- limit ili latencija - limit je broj okteta koji mogu čekati slobodne kredite, a latency je maksimalno vrijeme koje paket može čekati slobodni kredit unutar TBF-a
- burst/buffer/maxburst - veličina privremene memorije u oktetima - ovo je najveća vrijednost u oktetima za koju će biti slobodnih kredita
- mpu - Minimal packet Unit određuje najmanju veličinu kredita za jedan paket, potrebno iz razloga što paket veličine nula okteta zauzima ne manje od 64 okteta
pojasne širine

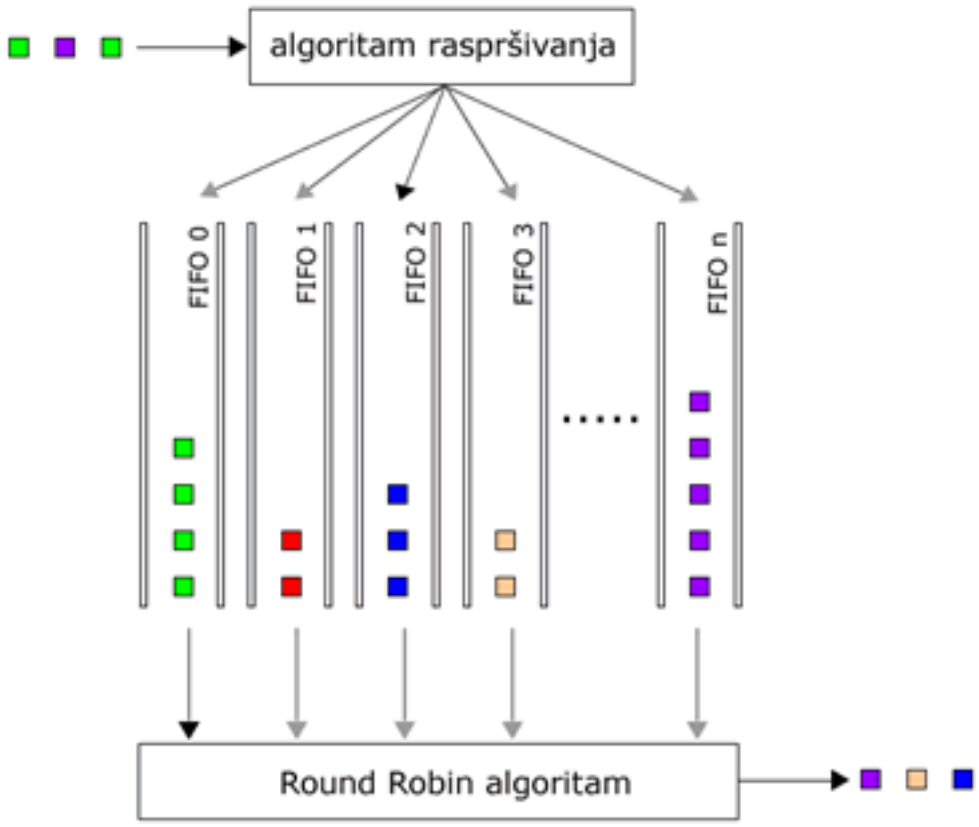
Ostali parametri su:
- rate - uzima se u obzir pri računanju maksimalnog vremena čekanja paketa
- peakrate - također se uzima u obzir pri izračunavanju maksimalnog vremena paketa, a služi pri određivanju brzine pražnjenja privremene memorije, maksimalno
1mbit/s zbog Linux ograničenja
- mtu/minburst - efektivno znači stvaranje nove privremene memorije, zbog ograničenja peakrate parametra

Primjer konfiguracije:

```
#tc qdics add dev ppp0 root tbf rate 200kbit latency 50ms burst 1540
```

Ovaj je primjer vrlo koristan u slučaju da se želi spojiti jedan mrežni uređaj s velikim redom (kao npr. DSL modem) na drugi, vrlo brzi mrežni uređaj (npr. mrežna kartica). U slučaju odlaznog prometa performanse uređaja (modema) drastično padaju, a razlog je taj što se red odlaznih paketa u modemu puni puno brže nego je uređaj sposoban poslati, a paketi koji pristižu ne mogu doći na red. Gornja linija smanjuje red odlaznih paketa, ali ne u samom uređaju (modemu), nego u jezgri operativnom sustava, tj. na mjestu gdje ga možemo kontrolirati. Brzinu od 200kbit treba zamijeniti stvarnom brzinom odlaznog prometa umanjenom za nekoliko postotaka.

Metoda stohastičnog pravednog reda
Metoda stohastičnog pravednog reda (Stohastic Fairness Queuing - SFQ) je jednostavna implementacija metode za upravljanje iz cijele obitelji ravnopravnih metoda za upravljanje. Manje je precizna od ostalih, ali je mnogo brža (potrebno je manje kalkulacija) uz zadržavanje jednakosti prema svim paketima.
Glavna karakteristika SFQ metode je tok koji se podudara sa TCP sesijom ili UDP tokom. Mrežni je promet podijeljen u velik broj FIFO redova, po jedan za svaki tok. Onda se on dalje preusmjerava tzv. Round Robin algoritmom, tj. svakom toku se naizmjenično daje jednaka šansa za slanje paketa. To rezultira vrlo ravnopravnim ponašanjem i isključuje mogućnost da jedan tok bude prioritetan. Ova metoda ne alocira red za svaki tok nego ima ugrađen algoritam raspršivanja koji raspodjeljuje promet preko konačnog broja redova. Međutim, kako je moguće da raspršivanje jedan tok bude više zastupljen od drugih, sami algoritam se mijenja često i nasumice i odatle naziv "stohastičan".
SFQ metoda je korisna samo u slučaju kad je mrežno sučelje stalno opterećeno do maksimuma jer, u suprotnom, nema reda i nema efekta koji proizlazi iz čekanja paketa i raspodjele njihovog redoslijeda. Ravnopravnost algoritma ovisi o broju redova i to na način da veći broj redova znači veću ravnopravnost. Shema SFQ metode je prikazana na slici 3.



Parametri SQF metode su:

- perturb - broj sekundi nakon kojeg će se promijeniti algoritam, preporučena vrijednost je 10 sekundi - ako se ne postavi, algoritam se nikad neće promijeniti
- quantum - količina okteta koja određuje koliko se tok može isprazniti prije nego sljedeći red počne s pražnjenjem
- limit - ukupni broj paketa koji mogu stati u red nakon kojeg ih SFQ počne brisati

Primjer konfiguracije:

```
# tc qdisc add dev ppp0 root sfq perturb 10
# tc –s –d qdisc ls qdisc sfq 800c: dec ppp0 quantum 1514b limit 12p flows 128/1024 p
erturb 10sec
```

Parametar 800c je automatski kreirana jedinstvena oznaka (handle), a limit 128 postavlja maksimalni broj paketa koji mogu čekati u redu. Ova konfiguracija je prikladna za slučaj da se želi postići ravnopravnost mrežnog prometa na računalu koje ima mrežno sučelje iste pojasne širine kao i mrežna veza na koju se spojeno, npr. obični modem.

Random Early Detection

Ova se metoda koristi pri upravljanju redovima čekanja na glavnim linkovima (backbone linkovima) koji imaju više od 100Mb pojasne širine
Normalni način rada usmjernika (routera) na internetu se zove tail-drop, a očituje se u stvaranju reda čekanja do neke veličine, a sav mrežni promet koji prelazi tu veličinu se briše. Taj je način vrlo nepravedan i može dovesti do retransmisijske sinkronizacije. To je pojava kad izbrisani mrežni promet, koji nije stao red čekanja usmjernika, uzrokuje zakašnjelu retransmisiju koja prepuni već zagušeni red čekanja. Da se se backbone usmjernici mogli nositi s kratkotrajnim zagušenjima, obično imaju implementirane velike redove čekanja. Međutim, iako su veliki redovi čekanja dobri za propusnost, mogu prilično povećati latenciju i uzrokovati praskovito ponašanje TCP protokola prilikom zagušenja.
Problemi s tail-drop-om na internetu se povećavaju jer raste upotreba aplikacija koje nisu "prijateljski raspoložene" prema mreži. Tu uskače Linux kernel koji nudi RED, Random Early Detection. Neki ga zovu i Random Early Drop jer taj naziv ukratko opisuje njegov način rada. Sami RED nije potpuno rješenje za cijeli tail-drop problem jer aplikacije koje loše implementiraju proces prilagođavanja retransmisije (backoff procedure) još uvijek mogu dobiti veći postotak pojasne širine, ali s RED metodom ne utječu toliko propusnost i latenciju ostalih konekcija. RED statistički briše pakete iz svih tokova podataka prije nego se red čekanja u potpunosti napuni. Na taj se način backbone link usporava na suptilniji način i sprječava se retransmisijska sinkronizacija. RED također pomaže TCP protokolu u pronalaženju pravedne brzine na način da se neki paketi brišu ranije, što drži red čekanja manjim i latenciju pod kontrolom. Vjerojatnost da se neki paket izbriše iz određene konekcije je proporcionalan pojasnoj širini koju ta konekcija koristi, a ne broju paketa koje šalje. RED je najbolji za upotrebu na backbone linkovima gdje se ne može priuštiti kompleksnost praćenja pojedine konekcije radi pravednog upravljanja redom čekanja.

Parametri kojima je moguće kontrolirati RED moetodu su:
- min - određuje minimalnu veličinu reda, izraženu u bajtovima, nakon koje će se paketi početi brisati
- max - određuje tzv. soft maximum, vrijednost koju će algoritam pokušati ne prijeći
- burst - određuje maksimalni broj paketa koji mogu proći u kratkotrajnom prekoračenju
- limit - vrijednost koja predstavlja sigurnosnu točku, iza koje će RED preći u tail-drop način rada
- avpkt - prosječna veličina paketa

Parametar min se može izračunati na način da se najveća prihvatljiva latencija pomnoži s dostupnom pojasnom širinom. Ako se parametar postavi na premalenu vrijednost, smanjit će se propusnost, a prevelika vrijednost će smanjiti latenciju. Parametar max bi trebao biti najmanje dva puta veći od min vrijednosti. Parametar burst određuje ponašanje RED algoritma u situacijama kratkotrajnih prekoračenja. On bi trebao biti veći od vrijednosti dobivene dijeljenjem min i avpkt parametara. Parametar limit se obično postavlja na vrijednost osam puta veću od max parametra, dok parametar avpkt postavljen na vrijednost 1000 radi uredno na linkovima koji imaju MTU od 1500 bajtova.

Zaključak

Sve besklasne metode su jednostavne metode koje upravljaju mrežnim prometom na način da ga

preraspodjeljuju, usporavaju (tj. unose kašnjenje) ili brišu. Ukratko:
- za jednostavno usporavanje mrežnom prometa upotrijebiti TBF metodu koja pouzdano radi na velikim pojasnim širinama ako se dobro odredi veličina privremene memorije (bucket)
- u slučaju da je link konstantno maksimalno opterećen, a želja je da sav mrežni promet bude ravnopravan, upotrijebiti SFQ metodu
- ako ste vlasnik backbone linka i znate što radite, upotrijebite RED
- ako se mrežni promet prosljeđuje, upotrijebiti TBF na sučelju na koje se promet proslijeđuje

Linkovi:
Linux Advanced Routing and Traffic Control [9]
RED Gateways for Congestion Avoidance [22]

- Logirajte [1] se za dodavanje komentara

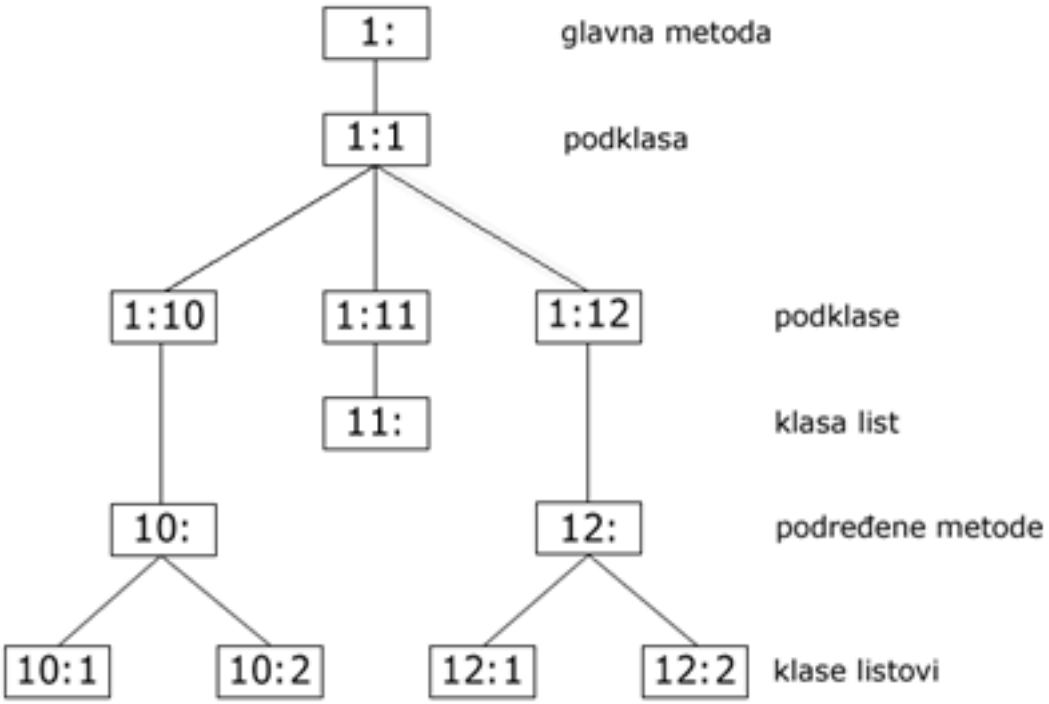
pet, 2007-11-30 13:19 - Mirko Lovričević**Kategorije:** Mreža [21]
Vote: 0

No votes yet

Filtriranje mrežnog prometa na aplikacijskoj razini pomoću Linux rješenja - 3.dio

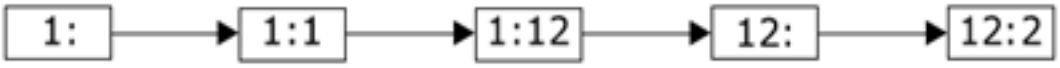
Kod Linux operativnih sustava kontrola mrežnog prometa (ulazak paketa u redove čekanja) je bazirana na klasama, tj. metoda upravljanja može sadržavati klase (eng. *classful qdisc*). Taj je princip vrlo koristan u slučaju da je promet na mrežnom sučelju različite vrste i da je svaku vrstu potrebno oblikovati na željeni način. Upotreba metoda koje koriste klase je vrlo fleksibilna i to na način da klase mogu sadržavati jednu ili više podređenih klasa ili jednu metodu za upravljanje. Dalje, podređene klase također mogu imati svoje podređene klase ili jednu metodu za upravljanje, što dovodi do moguće vrlo kompleksnog sustava za upravljanje mrežnim prometom. Klase koje nemaju svojih podređenih klasa zovu se klase listovi (eng. *leaf classes*) i one imaju dodijeljenu samo metodu za upravljanje. Također, klasi se može dodijeliti jedan ili više filtara kojima se omogućava usmjeravanje mrežnog prometa na određenu pod klasu ili se obavlja njegova selekcija (njegova ponovljena klasifikacija ili brisanje). Slučaj kad se mrežni promet usmjerava na neku od podređenih klasa se zove klasificiranje mrežnog prometa (eng. *classify*). Većina metoda koje koriste klase također omogućavaju i oblikovanje mrežnog prometa, što je vrlo korisno ako se želi kontrolirati brzina i raspoređivati redoslijed toka (npr. pomoću metode SFQ). Oblikovanje se najčešće koristi u slučaju da je izlazno mrežno sučelje velike brzine (npr. mrežna kartica) spojeno na mrežno sučelje male brzine (npr. kabelski modem). Tada se brzina izlaznog sučelja smanjuje na brzinu ulaznog sučelja i ne dolazi do zagušenja mrežnog prometa ili prioritiziranja nekog toka.

Svako mrežno sučelje ima jednu metodu za upravljanjem redom (eng. *root qdisc*) i unaprijed definirano, to je već opisana metoda pfifo_fast. Svakoj metodi i klasi se dodjeljuje jedinstvena oznaka (eng. *handle*) preko koje se u kasnijoj konfiguraciji pristupa toj metodi tj. klasi. Jedinstvene oznake se sastoje od dva dijela tj. broja, *major* i *minor* koji tvore jedinstvenu oznaku na sljedeći način: *<major>:<minor>*. Običaj je da se glavna metoda označi s oznakom '1'. Kako sve metode imaju *minor* broj '0', onda puna oznaka glavne metode glasi '1:0'. Klase imaju isti *major* broj kao i njihovi nadređeni objekt (metoda ili klasa). Tako *major* broj mora biti jedinstven za ulazni tj. izlazni red, a *minor* brojevi jedinstveni unutar metode i njenih klasa. Tipična hijerarhija može izgledati kao na slici 1.



Važno je naglasiti da paketi ulaze u red i izlaze iz njega samo u glavnoj metodi, to je jedini dio ovog stablastog prikaza s kojim komunicira jezgra operativnog sustava. Paketi ulaze u stablo na vrhu i onda se klasificiraju prema dnu da bi zatim izlazili prema vrhu redoslijedom određenim u samim metodama i njima pridruženim filtrima.

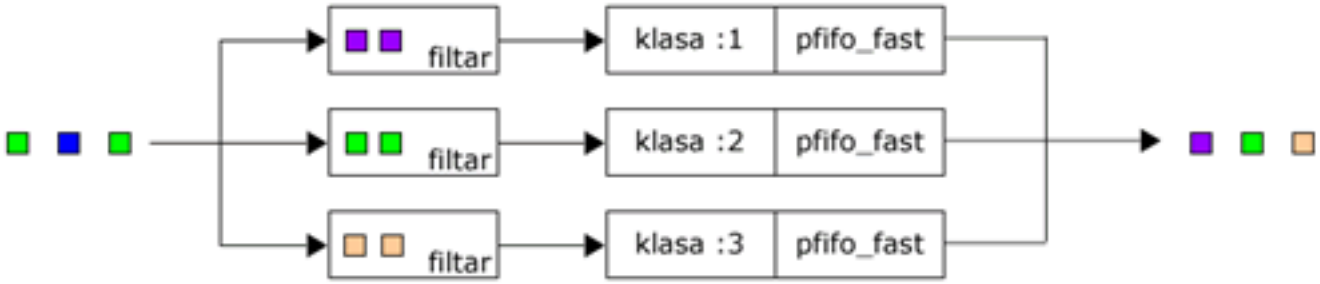
Npr. paket se može klasificirati u lancu na način prikazan na slici 2.



Ovdje je paket dodijeljen metodi u klasi 12:2. U ovom primjeru svaka grana stablastog prikaza ima svoj filtar i paket je postupno klasificiran. Također je moguće da paket bude klasificiran na drugi način, prikazan na slici 3.

U ovom se slučaju paket iz glavne metode usmjerava direktno u klasu 12:2, ali ovaj način je manje fleksibilan zbog nepostojanja postupnog klasificiranja.

Prio metoda za upravljanje
Ova metoda ne oblikuje mrežni promet nego ga samo klasificira na osnovu parametara definiranim u filtrima. Slična je besklasnoj metodi pfifo_fast s razlikom što umjesto tri podređena reda, unaprijed definirano ima tri klase koje sadrže FIFO metodu bez unutrašnje strukture, ali moguće ih je zamijeniti bilo kojom metodom. Ova je metoda korisna u slučaju da se određenom dijelu mrežnog prometa želi dodijeliti veći prioritet ne koristeći samo TOS polja u zaglavlju paketa, nego upotrebom svih prednosti klasa. Upravo zbog razloga što ne oblikuje mrežni promet, kao i metoda SFQ, ova je klasna metoda najbolja u slučaju kad je mrežno sučelje potpuno opterećeno ili se koristi unutar metode koja oblikuje mrežni promet. Shema metode prio je prikazana na slici 4.

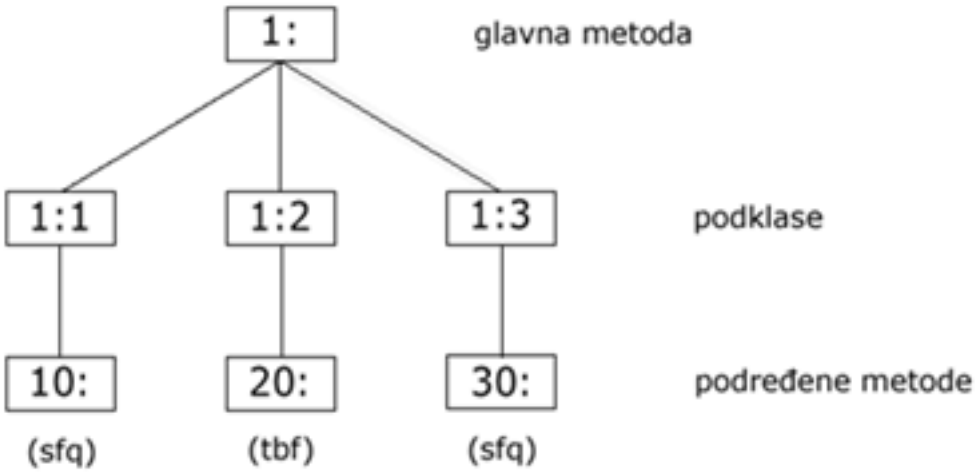


Prilikom napuštanja reda prvo se prazni podređena klasa 1. Kad se ona isprazni, podređena klasa 2 dolazi na red i na kraju se prazni podređena klasa 3.

Parametri dostupni uz prio metodu su:

- *bands* – broj klasa tj. filtara koji se kreiraju
- *priomap* – parametar usko povezan s *band* parametrom, ista funkcija kao i kod metode pfifo_fast

Za primjer konfiguracije kreirat ćemo stablo prikazano na slici 5.



Neklasificirani mrežni promet ćemo usmjeriti na klasu 30, a interaktivni mrežni promet na klase 10 i 20.

```
# tc qdisc add dev eth0 root handle 1: prio
## sljedece tri linije kreiraju klase 1:1, 1:2 i 1:3
# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600 limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

Class Based Queuing (CBQ)
Ovo je najkompleksnija metoda koja postoji i najteža za ispravno konfiguriranje. Razlog tome je nepreciznost algoritma koji ne prati najbolje rad Linux operativnog sustava. Ova metoda, osim što koristi klase, oblikuje mrežni promet, ali to radi dobro samo u određenim slučajevima. Metoda CBQ radi na način da osigurava dovoljnu neopterećenost mrežne veze i time se brzina mrežnog prometa svodi na konfiguriranu vrijednost. To se radi na način da metoda izračunava vrijeme koje bi trebalo proći između prosječnih paketa. Tijekom rada mjeri se efektivno vrijeme praznog hoda (vrijeme kad nema paketa koji pristižu na mrežno sučelje) i to pomoću EWMA (*Exponential Weighted Moving Average*) metode koja pretpostavlja da je zadnji paket koji je stigao na mrežno sučelje

eksponencijalno važniji od prethodnog paketa. Izračunato vrijeme praznog hoda se oduzima od izmjerenog i dobiveni broj se zove *avgidle*. Kod savršeno izbalansirane mrežne veze *avgidle* iznosi nula, što znači da paket stigne točno u izračunatom intervalu. Kako je vrlo teško precizno izmjeriti vrijeme praznog hoda, CBQ ponekad potpuno promaši zadane vrijednost. Razlog više mogu biti različiti problemi poput loše implementiranog upravljačkog programa (*drivera*) mrežne kartice, nemogućnost sabirnice da propusti određenu količinu mrežnog prometa i slično.
Neopterećena mrežna veza uzrokuje vrlo velik pozitivan *avgidle*, što bi nakon dužeg vremena praznog hoda omogućilo neograničenu mrežnu propusnost. Da bi se to spriječilo, koristi se *maxidle* parametar.
S druge strane, mrežna veza koja je preopterećena ima negativan *avgidle* i ako on postane previše negativan, CBQ prestaje s radom na određeni period i tada se nalazi u *overlimit* stanju. Teoretski, u tom bi se slučaju CBQ trebala zagušiti točno onoliko vremena koliko je izračunato vrijeme između dva paketa, zatim propustiti jedan paket, ponovo ući u *overlimit* stanje i tako sve dok zagušenje ne prođe. U tu se svrhu koristi *minburst* parametar.
Parametri dostupni uz metodu CBQ su:

- *avpkt* – prosječna veličina paketa u oktetima, potrebna za izračunavanje *maxidle* parametra
- *bandwidth* – fizička pojasna širina mrežne veze, potreban za izračunavanje vremena praznog hoda
- *cell* – vrijeme potrebno da se paket pošalje može rasti u koracima, ovisno o veličini paketa. Npr. paketima veličine 800 i 806 okteta treba isto vremena da se pošalju. Ovaj parametar određuje granularnost. Obično se postavlja na vrijednost 8, a mora biti potencija broja 2
- *maxburst* – u slučaju da *avgidle* broj ima vrijednost *maxidle*, ovoliki broj paketa će biti propušten prije nego se *avgidle* vrati na nulu. Vrijednost parametra *maxidle* je jedino moguće postaviti pomoću ovog parametra
- *minburst* – u slučaju *overlimit* stanja idealno je propustiti jedan paket unutar točno izračunatog vremena praznog hoda. Međutim, kako je kod Unix jezgri vrlo teško raspoređivati događaje unutar vremena od 10ms, bolja je situacija biti u zagušenju duže vremena, zatim propustiti *minburst* broj paketa pa onda ući u stanje praznog hoda isto toliko vremena
- *minidle* – vrijednost na koju se postavlja *avgidle* ukoliko postane premalen (da bi se spriječilo ispadanje mrežne veze)
- *mpu* – na ethernetu paketi veličine nula okteta se postavljaju na veličinu od 64 okteta i kao takvi uzimaju vrijeme potrebno za slanje. Ovaj parametar služi za precizno izračunavanje vremena praznog hoda
- *rate* – parametar brzine

Osim samog oblikovanja mrežnog prometa, metoda CBQ radi istim načinom kao i klasna metoda prio, što znači da klase unutar metoda imaju prioritete. Svaki put kad se zatraži paket, započinje težinski Round Robin proces (*Weighted Round Robin - WRR*) i to s klasom najvećeg prioriteta. Klase se grupiraju i ispituju da li u njima postoje paketi koji čekaju. Ako postoje, onda se te klase prve obrađuju. Parametri koji su dostupni uz WRR proces su:

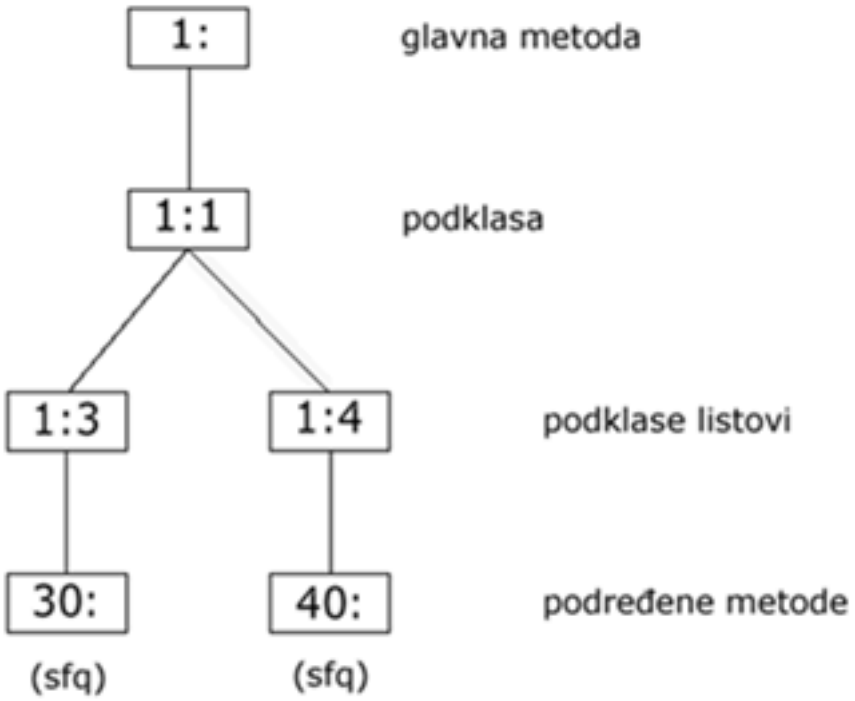
- *allot* – prilikom obrade klasa po prioritetima svaka klasa može poslati određenu količinu podataka. Ovaj parametar je osnovna jedinica te veličine, a koristi se zajedno s *weight* parametrom
- *prio* – parametar koji određuje prioritet
- *weight* - parametar koji se množi s parametrom *allot* da bi se dobila količina podataka koju svaka klasa može poslati kad dođe na red

U svakoj metodi CBQ postoje mehanizmi koji omogućavaju fino podešavanje rada. Tako se npr. ne ispituju klase u kojima nema paketa koji čekaju, a klasama koje su u *overlimit* stanju se smanjuje prioritet.

Jedna od naprednijih mogućnosti metode CBQ jest mehanizam posuđivanja pojasne širine (eng. *bandwidth*). Moguće je specificirati neku klasu koja će posuditi pojasnu širinu od neke druge klase, kao i da jedna klasa posudi dio svoje pojasne širine drugoj klasi. Parametri dostupni uz ove mogućnosti su:

- *isolated/sharing* – klasa koja je konfigurirana kao *isolated* neće posuditi svoju rezerviranu pojasnu širinu nekoj drugoj klasi. Obratno, klasa kojoj je postavljen parametar *sharing* će posuditi svoju pojasnu širinu
- *bounded/borrow* – klasa kojoj je postavljen parametar *bounded* neće posuđivati pojasnu širinu od neke druge klase. Obratno, klasa koja je konfigurirana kao *borrow* će pokušati posuditi pojasnu širinu od druge klase

Tipična situacija bi bila da na jednoj mrežnoj vezi postoje dvije klase od kojih je svaka i *isolated* i *bounded*. Na taj se način osigurava limitiranost svake klase na postavljene vrijednosti. Također je moguće da unutar neke klase koja nema mogućnost dijeljenja pojasne širine postoje druge klase koje mogu dijeliti svoju pojasnu širinu.
Slijedi primjer konfiguracije koja limitira mrežni promet web poslužitelja na 5Mb, SMTP mrežni promet na 3Mb, a zajedno ukupno ne mogu preći 6Mb. Mrežna kartica je pojasne širine 100Mb i klase mogu posuđivati pojasnu širinu jedna od druge. Shema primjera prikazana je na slici 6



Sljedeće dvije linije kreiraju glavnu metodu unutar reda (eng. *root qdisc*) i podređenu klasu 1:1. ta je podređena klasa postavljena na *bounded*, što znači da ukupni mrežni promet ne može preći zadanu vrijednost, u ovom slučaju 6Mb.

```
# tc qdisc add dev eth0 root handle 1:0 cbq bandwidth 100Mbit avpkt 1000 cell 8
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 100Mbit rate 6Mbit weigh
t 0.6Mbit
   prio 8 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded
```

Sad se kreiraju dvije klase listovi koje su pomoću *weight* parametra ograničene na određene vrijednosti pojasne širine koju mogu koristiti, ali, kako su vezane za klasu 1:1, njihov zbroj nikad neće biti veći od maksimalne vrijednosti dostupne pojasne širine koja je postavljena za klasu 1:1.

```
# tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth 100Mbit rate 5Mbit
   weight 0.5Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000
# tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth 100Mbit rate 3Mbit
   weight 0.3Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000
```

Obje podređene klase imaju FIFO kao unaprijed definiranu metodu i ovdje se postavlja metoda SFQ da bi se osigurala jednakost obje vrste mrežnog prometa.

```
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
# tc qdisc add dev eth0 parent 1:4 handle 40: sfq
```

Sljedeće dvije linije se odnose na glavnu metodu za upravljanje i šalju mrežni promet s priključnih točaka 25 i 80 na prethodno kreirane podređene klase.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 80 0xffff f
lowid 1:3
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 25 0xffff f
lowid 1:4
```

Ako obje vrste mrežnog prometa pokušaju preći ukupnih 6Mb, pojasna širina će se podijeliti prema *weight* parametru postavljenom prilikom kreiranja klasa, što znači da će 5/8 pojasne širine biti rezervirano za web poslužitelj, a 3/8 za SMTP mrežni promet.
U ovom slučaju sav ostali promet nije klasificiran, što znači da će moći iskoristiti svu preostalu pojasnu širinu.

Hierarchical Token Bucket (HTB)
Nakon spoznaje kompleksnosti metode CBQ i obzirom da ona nije optimizirana na mnoge tipične situacije, stvorena je metoda HTB. Ona koristi princip metode TBF (privremena memorija koja se puni imaginarnim podacima) vezan za klase, i filtara uz korištenje mehanizama posuđivanja (iz metode CBQ) i kao takva omogućava vrlo precizno kontroliranje i oblikovanje mrežnog prometa. Mehanizam posuđivanja se odnosi na imaginarne podatke – kredite za slanje (eng. *tokens*) i to na način da podređene klase posuđuju kredite od svojih nadređenih objekata u slučaju da su prešli svoju zadani parametar brzine slanja paketa. Posuđivanje kredita i slanje paketa se nastavlja dok se ne dosegne limit postavljen od strane korisnika. U tom će trenutnu podređena klasa početi stavljati pakete u stanje čekanja sve dok još kredita ne postane dostupno.
Može se reći da metoda HTB osigurava da je pojasna širina koja se daje nekoj klasi na korištenje najmanje onolika koliko je ta klasa zatražila i koliko je za nju rezervirano. Ova se metoda pokazala najbolja u slučajevima kada postoji određena pojasna širina koja se želi podijeliti na način da svaki njen dio ima zagarantiranu vrijednost uz istovremeno zadržavanje mogućnosti posuđivanja pojasne širine za slučaj da se koristi manje pojasne širine od rezervirane.
Parametara dostupnih uz ovu metodu ima samo nekoliko:

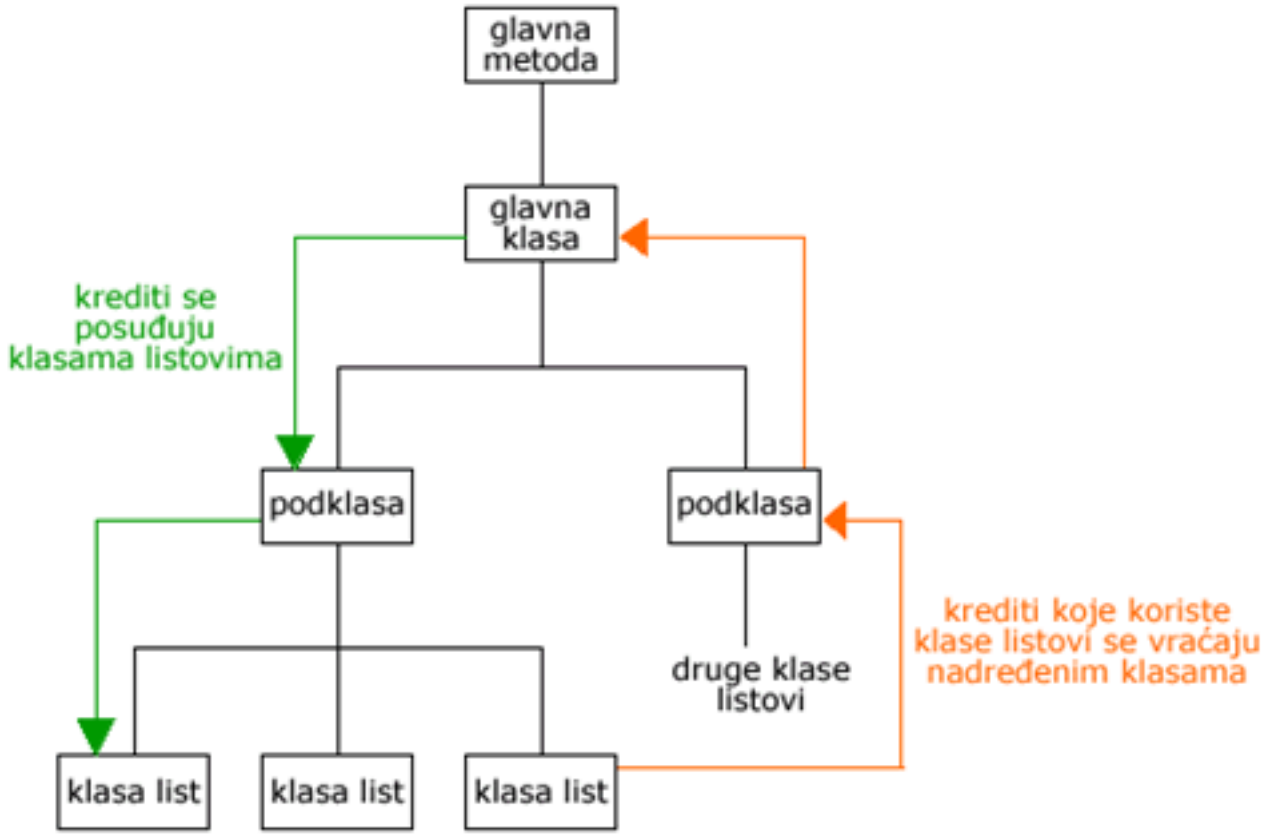
- *rate* – parametar brzine, određuje pojasnu širinu koju klasa može koristiti
- *ceil* – parametar koji određuje maksimalnu pojasnu širinu za pojedinu klasu, što limitira pojasnu širinu koju klasa može posuditi od druge klase. Unaprijed definirana vrijednost je ista kao i postavljeni *rate* parametar, a njegova vrijednost se može nalaziti između veličine parametra *rate* te klase te veličine *ceil* parametra svojih podređenih klasa
- *burst/cburst* – parametar koji određuje količinu podataka koja može biti poslana maksimalnom brzinom koju dozvoljava sklopovlje bez posluživanja neke druge klase. Također treba biti najmanje velik koliko i vrijednost *burst/cburst* parametra svake podređene klase
- *prio* – parametar koji određuje prioritet na način da klase s višim prioritetom prve dobivaju šansu za posuđivanje pojasne širine
- *quantum* – ključni parametar za kontrolu mehanizma posuđivanja. Njega "po defaultu" određuje sama metoda HTB, a služi za podjelu mrežnog prometa između podređenih klasa (količinom većom od *rate*, a manjom od *ceil*) i slanje paketa iz tih istih klasa

Kako postoje samo dvije primarne klase koje se mogu kreirati pomoću metode HTB, tablica 1 opisuje moguća stanja mehanizma za posuđivanje, a slika 7 ponašanje mehanizma za posuđivanje.

Tip klase	Stanje klase	Podcsetа akcija
list	<rate	klasa će poslati onoliko paketa koliko ima dostupnih kredita

list	>rate, <ceil	klasa će pokušati posuditi kredite od nadređenog objekta, ako postoje slobodni krediti, oni će biti posuđeni u quantum količini i podređena će klasa poslati onoliko podataka koliko je određeno burst parametrom
list	>ceil	nijedan paket neće biti poslan, što dovodi do kašnjenja
podređena klasa, glavna klasa	<rate	nadređena klasa će posuditi kredite svojim podređenim objektima
podređena klasa, glavna klasa	>rate, <ceil	klasa će pokušati posuditi kredite odvoga nadređenog objekta i onda ih proslijediti svojim podređenim objektima ukoliko oni to zatraže
podređena klasa,glavna klasa	>ceil	klasa neće pokušati posuditi kredite od svog nadređenog objekta i, prema tome, neće ih proslijediti svojim podređenim objektima

Slika 7.



Primjer konfiguracije je gotovo isti kao i kod metode CBQ.

```
# tc qdisc add dev eth0 root handle 1: htb default 30
# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
```

Preporuka je da se za podređene klase koristi metoda SFQ:

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Na kraju se dodaju filtri koji usmjeravaju mrežni promet na podređene klase 1:10 i 1:20.

```
# U32="tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip dport 80 0xffff flowid 1:10
# $U32 match ip sport 25 0xffff flowid 1:20
```

Iz priloženog se vidi da je ova konfiguracija jednostavnija od one korištene s metodom CBQ, uz isti rezultat. Klase 10 i 20 imaju svoju rezerviranu pojasnu širinu i ako im je potrebno više, posuđivanje se obavlja u odnosu 5:3. Neklasificirani mrežni promet se usmjerava na klasu 30 koja može posuditi svu preostalu pojasnu širinu, a korištenjem metode SFQ dobivena je ravnomjerna raspodjela mrežnog prometa.

Zaključak
Metode za upravljanje redovima koje koriste klase omogućavaju da se na jednom fizičkom linku simulira više sporijih linkova te se preko njih na različite načine šalju različite vrste mrežnog prometa. Njihove napredne mogućnosti dovele su do njihove integracije u Linux kernel (metoda CBQ od kernel verzije 2.1, a metoda HTB od kernel verzije 2.4). Metodu CBQ će preferirati korisnici koji još uvijek koriste starije verzije kernela. Međutim, iz razloga što ta metoda i oblikuje mrežni promet, a zbog kompleksnosti algoritma to ne radi na adekvatan način, preporuka je da se pređe na metodu HTB koja ne ovisi o karakteristikama mrežnog sučelja. Njena manje kompleksna konfiguracija, brzina te odlična dokumentacija čine je odličnim izborom za klasificiranje i oblikovanje mrežnog prometa.

čet, 2008-07-24 14:30 - Mirko Lovričević**Kategorije:** Mreža [21]
Vote: 0

No votes yet

Source URL: https://sysportal.carnet.hr/node/521

Links
[1] https://sysportal.carnet.hr/sysportallogin
[2] http://www.netfilter.org/
[3] http://www.snowman.net/projects/ipt_recent
[4] https://sysportal.carnet.hr/taxonomy/term/17
[5] https://sysportal.carnet.hr/taxonomy/term/30
[6] https://sysportal.carnet.hr/taxonomy/term/25
[7] https://sysportal.carnet.hr/taxonomy/term/26
[8] https://sysportal.carnet.hr/taxonomy/term/28
[9] http://lartc.org
[10] https://kernel.org
[11] http://netfilter.org
[12] http://freshmeat.net/projects/iproute2
[13] http://sourceforge.net/project/showfiles.php?group_id=80085%20-%20netfilter-layer7-vX.Y.tar.gz
[14] http://sourceforge.net/project/showfiles.php?group_id=80085%20-%20l7-protocols-YYYY-MM-DD.tar.gz
[15] https://sysportal.carnet.hr/system/files/bridges.txt
[16] https://sysportal.carnet.hr/system/files/non-bridges.txt

[17] https://sysportal.carnet.hr/system/files/nat.txt
[18] http://l7-filter.sourceforge.net/
[19] http://l7-filter.sourceforge.net/protocols
[20] https://sysportal.carnet.hr/taxonomy/term/22
[21] https://sysportal.carnet.hr/taxonomy/term/29
[22] http://www.icir.org/floyd/papers/red/